COLOMBANI Eric
3ème Année
Option CDSE



Projet de Fin d'Etudes

« Programmation des asservissements et de la communication CAN sur le Cycab»



Stage ingénieur

Date : du 01/03/2000 au 30/06/2000
Lieu : Institut National de Recherche en Informatique et Automatique à Montbonnot (38) (I.N.R.I.A Rhône Alpes).
Maître de stage : M. Hervé MATHIEU, ingénieur de recherche aux moyens robotiques à l'I.N.R.I.A Rhône Alpes.
Tuteur de stage : M. Jocelyn CHANUSSOT.

REMERCIMENTS :

Je tiens à remercier vivement l'INRIA Rhône-Alpes de m'avoir accueilli pour effectuer mon stage de fin d'études dans le service robotique, m'offrant ainsi la possibilités d'acquérir une expérience professionnelle très enrichissante.

Je remercie ensuite tout particulièrement Hervé Mathieu, mon responsable de stage, pour ses conseils et le temps qu'il a bien voulu me consacrer tout au long de ce stage. Il m'a patiemment expliqué les concepts que je ne connaissais pas. C'est grâce a lui et au travail proposé que ce stage a été si formateur.

Je souhaite également remercier toute l'équipe du Service Robotique, Vision et Réalité Virtuelle de l'INRIA Rhône-Alpes : Gérard Baille, Soraya Arias et Jean-François Cuniberto pour leur aide lorsque certains problèmes se sont présentés.

Je remercie aussi tous les stagiaires présents sur place dans la halle robotique de l'INRIA où se déroulait mon stage.

Table des matières :

1 - Description de l'entreprise :

1-1 PRESENTATION DE L'INRIA :

a) Introduction :

Créé en 1967 à Rocquencourt près de Paris, l'INRIA, Institut National de Recherche en Informatique et en Automatique, est un établissement public à caractère scientifique et technologique qui mène des recherches avancées dans le domaine des sciences et technologies de l'information et de la communication. Ce domaine inclut l'informatique et l'automatique, mais aussi les télécommunications et le multimédia, la robotique, le traitement du signal et le calcul scientifique. L'INRIA est placé sous la double tutelle du Ministère de la Recherche et du Ministère de l'Economie, des Finances et de l'Industrie.

L'INRIA a l'ambition d'être au plan mondial un institut de recherche au cœur de la société de l'information.

Sa volonté est de mettre en réseau des compétences et des talents de l'ensemble du dispositif de recherche français dans le domaine des STIC. Ce réseau permet de mettre l'excellence scientifique au service des progrès technologiques, créateurs d'emplois, de richesse et de nouveaux usages répondant à des besoins socio-économiques.

Son organisation décentralisée (5 unités de recherche), ses petites équipes autonomes et évaluées régulièrement permettent à l'INRIA d'amplifier ses partenariats, 47 projets de recherche sur 87 sont communs avec les universités, les grandes écoles et les organismes de recherche. Il renforce son implication dans les travaux de valorisation des résultats de recherche et le transfert technologique : 600 contrats R&D avec l'industrie et un peu moins d'une cinquantaine de sociétés sont issues de l'INRIA.

b) Quelques chiffres (décembre 2000) :

- *Ressources budgétaires :* - dotation de l'état : 442 MF HT
  - ressources propres : 174 MF HT
- *Ressources humaines :* - titulaires INRIA : 724.
  - post-Doctorants et Contractuels : 256.
  - doctorants : 550.
  - chercheurs et enseignants d'autres organismes : 230.
  - conseillers, collaborateurs divers et invités : 430.
- *Indicateurs :* - contrats de recettes actifs : plus de 600.
  - contrats de recettes signés dans l'année : plus de 200.
  - un peu moins d'une cinquantaine de sociétés sont issues de l'INRIA, depuis Ilog, aujourd'hui cotée au Nasdacq, jusqu'aux toutes dernières, 5 en 1998, 6 en 1999, 11 en 2000.

- 7 brevets initiaux déposés en 1999 : 1 est en pleine propriété INRIA, les autres sont en copropriétés, 5 avec des industriels et 1 avec une université.

c) Gestion des projets :

Un projet de recherche est une équipe rassemblant de 15 à 20 personnes autour d'une thématique forte et sur des objectifs scientifiques précis. Ces équipes gèrent de façon autonome leur budget. Les résultats obtenus et les retombées industrielles qu'ils induisent, sont régulièrement évalués.

Une action de recherche est un groupe de chercheurs poursuivant un travail commun sur une thématique spécifique qui peut aboutir à la création d'un projet de recherche.

1-2 PRESENTATION DE L'INRIA RHONE-ALPES :

a) Introduction :

Créée en décembre 1992, l'INRIA Rhône-Alpes est la plus récente des cinq unités de recherche de l'INRIA.

Menées au sein d'une région en plein essor technologique, les activités de l'unité de recherche INRIA Rhône-Alpes mobilisent plus de 330 personnes, dont 220 chercheurs, géographiquement réparties sur trois sites : le site de l'INRIA à Montbonnot, le campus universitaire de Grenoble et l'Ecole Normale Supérieure de Lyon.

Ces activités s'inscrivent dans le cadre des missions que doit accomplir l'INRIA en tant qu'établissement de recherche national, tout en se focalisant sur les objectifs stratégiques poursuivis par l'institut dans le domaine des sciences et technologies de l'information et de la communication.

L'INRIA Rhône-Alpes accueille plus de 130 doctorants, ingénieurs et stagiaires. Ses chercheurs participent à l'enseignement supérieur au sein des universités et grandes écoles de la région Rhône-Alpes (Institut National Polytechnique de Grenoble, université Joseph Fourier, université Pierre Mendès-France, université de Savoie, Ecole Nationale Supérieure de Lyon).

b) Missions de l'INRIA Rhône-Alpes :

En tant qu'unité de recherche de l'INRIA, les principales missions de l'INRIA Rhône-Alpes sont, selon le décret du 2 août 1985 portant sur l'organisation et le fonctionnement de l'institut :

- *Entreprendre des recherches fondamentales et appliquées.*
- *Réaliser des systèmes expérimentaux.*
- *Organiser des échanges scientifiques internationaux.*

- *Assurer le transfert et la diffusion des connaissances et du savoir-faire.*
- *Contribuer à la valorisation des résultats de la recherche.*
- *Contribuer, notamment par la formation, à des programmes de coopération pour le développement.*
- *Effectuer des expertises scientifiques.*
- *Contribuer à des actions de normalisation.*

c) Pôles de recherche :

L'INRIA Rhône-Alpes mène ses activités en étroite collaboration avec les laboratoires de recherche publics et privés, nationaux et internationaux, et elle entretient des liens privilégiés avec l'institut d'Informatique et Mathématiques Appliquées de Grenoble (IMAG). Ces activités sont organisées autour de quatre pôles de recherche :

- *Maîtriser les systèmes et réseaux informatiques* : Réseaux, parallélisme et systèmes répartis.
- *Aider à la conception et à la création* : Bases de connaissances, documents multimédia, modèles cognitifs.
- *Percevoir, simuler et agir* : Synthèse d'images, réalité virtuelle, vision par ordinateur et robotique.
- *Modéliser les phénomènes complexes* : Automatique, simulation et calcul scientifique.

1-3 PRESENTATION DU SERVICE :

Mon stage se déroule au sein du service robotique, vision et réalité virtuelle (RV2) de l'INRIA Rhône-Alpes dont le rôle est la mise en œuvre des outils matériels et logiciels pour les expérimentations robotiques des projets de recherche du site.

Ce service compte :

- 3 ingénieurs de recherche,
- 1 ingénieur expert,
- 1 technicien,
- 1 assistante de service.

a) Les missions du service :

Les missions qui lui sont attribuées sont de trois types :

- *Activité de service :*
  - ✓ maintenance des systèmes robotiques.
  - ✓ installation et maintenance de logiciels spécialisés.
  - ✓ interface entre les utilisateurs et le service informatique.
  - ✓ assistance aux utilisateurs.
- *Activité de développement :*
  - ✓ mise en place d'expérimentations.
  - ✓ développement de logiciels dédiés à la robotique.
- *Activité de recherche :*

- ✓ conception de systèmes robotiques.
- ✓ confrontation théorie et expérimentation.

Le but du Service Robotique est de fédérer l'effort expérimental en favorisant :

- *les expérimentations inter-projets,*
- *la mise en commun des moyens expérimentaux,*
- *les outils réutilisables (environnement de développement, machine de vision...).*

b) Les projets concernés :

Les moyens robotiques travaillent avec les projets tels que "*Interaction homme-machine, images, données, connaissances*" et "*Simulation et optimisation de systèmes complexes*" impliqués en robotique et vision.

- *SHARP* : Programmation automatique et systèmes décisionnels en robotique
- *MOVI* : Modélisation, localisation, reconnaissance et interprétation en vision par ordinateur
- *BIP* : Conception et contrôle de robots marcheurs et applications.
- *iMAGIS* : Modèles, algorithmes, géométrie pour le graphique et l'image de synthèse.
- *PRIMA* : développer des techniques pour l'intégration de la perception et de l'action en robotique.

2 Présentation du stage :

2-1 THEME DU STAGE : LE CYCAB :



Figure 1 – Le Cycab

Dans le cadre de la route automatisée, l'INRIA a imaginé un système de transport original de véhicules en libre-service pour la ville de demain. Ce système de transport public est basé sur une flotte de petits véhicules électriques spécifiquement conçus pour les zones où la circulation automobile doit être fortement restreinte. Pour tester et illustrer ce système, un prototype, nommé Cycab (**Figure 1**), a été réalisé. Aujourd'hui l'INRIA Rhône-Alpes possède trois Cycabs pour développer des applications autour de ce mode de transport.

2-2 L'ETAT D'AVANCEMENT DU PROJET :

Les chercheurs de l'INRIA et de l'Inrets (Institut National de Recherche sur les Transports et leur Sécurité) travaillent depuis 1991 sur de nouveaux moyens de transport intelligent pour la ville. Ils étudient en particulier le concept du libre-service et celui de la voiture automatique. Les premiers résultats de recherche ont débouché sur le projet Praxitèle (1993-1999), qui était en exploitation à Saint-Quentin-en-Yvelines. Les partenaires industriels du projet étaient CGFTE (la filiale transports publics de Vivendi), Dassault Electronique, EDF et Renault.

Dans le cadre du projet Praxitèle l'INRIA a démontré la faisabilité de la conduite automatique sous certaines conditions : créneau et train de véhicule expérimenté sur une Ligier électrique instrumentée à cet effet.

Pour des raisons de législation et de responsabilité ces automatismes de conduite n'ont pas pu être implémentés sur les Clio électriques de Saint-Quentin-en-Yvelines. Le Cycab a ensuite été développé par l'INRIA avec l'aide de l'Inrets, de EDF, de la RATP et de la société Andruet S.A. pour montrer le potentiel de l'informatique dans la conduite de véhicules. Le Cycab est un véhicule électrique à quatre roues motrices et directrices avec une motorisation indépendante pour chacune des roues et pour la direction. Pour contrôler et commander les 9 moteurs du Cycab (4 de traction, 1 de direction et 4 de frein), une architecture matérielle a été choisie. Elle est constituée de nœuds intelligents pouvant gérer les différents moteurs du Cycab et répartie autour d'un bus de terrain CAN (Controller Area Network), très répandu dans le monde de l'automobile.

Le rôle des nœuds est d'asservir les moteurs en fonction des consignes de vitesse et de braquage qui transitent sur le bus CAN soit en provenance de l'interface homme-machine (la position du joystick), soit par un programme de planification de trajectoires. Le nœud doit donc non seulement être capable de fournir la puissance nécessaire aux moteurs, mais aussi exécuter les boucles d'asservissement de vitesse ou de position. Pour ce faire il doit prendre en compte un certain nombre d'informations en provenance des capteurs proprioceptifs : état, odométrie, fins de course, mesures de température, de courant, …

Cette description succincte fait apparaître trois entités composant un nœud :

- Un **module de puissance** avec des transistors MOS-FET de puissance et leur commande de GATE pour piloter les moteurs.

- Un **module d'interface** et de communication dont le rôle est essentiellement de mettre en forme les signaux Tout ou Rien ou de convertir les signaux analogiques en provenance des capteurs, du joystick ou allant vers les indicateurs d'état pour qu'ils soient exploitables par le micro-contrôleur. Ce module gère aussi les communications sur le bus CAN.

- Un **module de calcul** qui, à partir des consignes et des données proprioceptives, calcule les courants à envoyer aux moteurs.

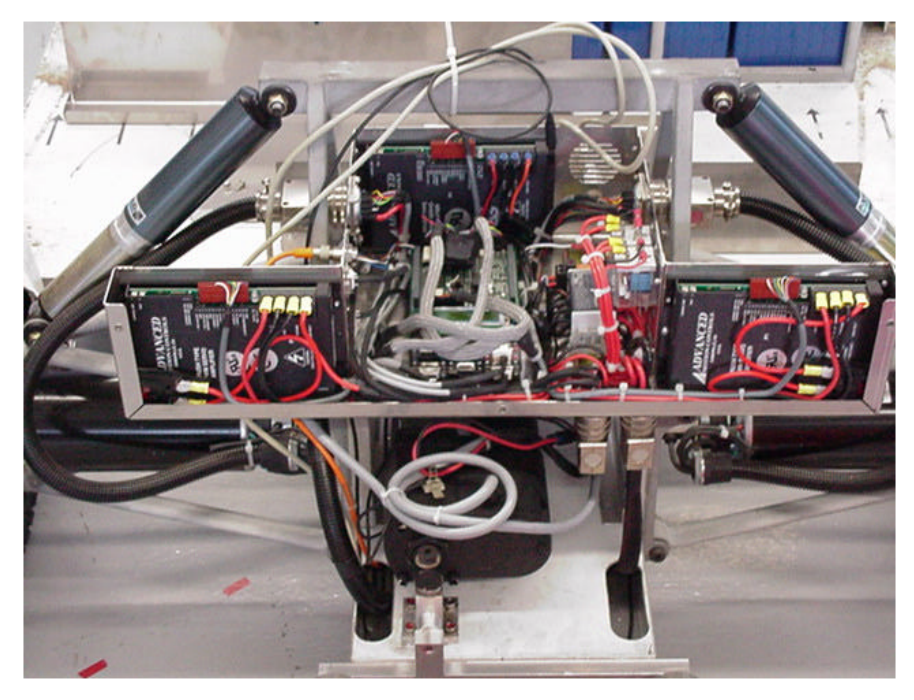


Figure 2 – Nœud avant

Dans une première version, des nœuds basés sur le micro-contrôleur MC68332 de chez Motorola (cœur 68020) étaient connectés sur ce bus. Le Cycab a ensuite évolué, et les nœuds ont été remplacés par de nouvelles cartes à base de micro-contrôleur MPC555 32 bits (cœur Power-PC) construites par la société Virtual Micro Design. Le Cycab est maintenant commercialisé par la société Robosoft. On peut voir sur la **figure 2** le nœud placé à l'avant du Cycab autour de trois amplificateurs de puissance (commande de deux moteurs de traction et d'un moteur de direction). La **figure 3** présente un agrandissement de la partie "intelligence", et **figure 4** présente les différents éléments sur la carte.

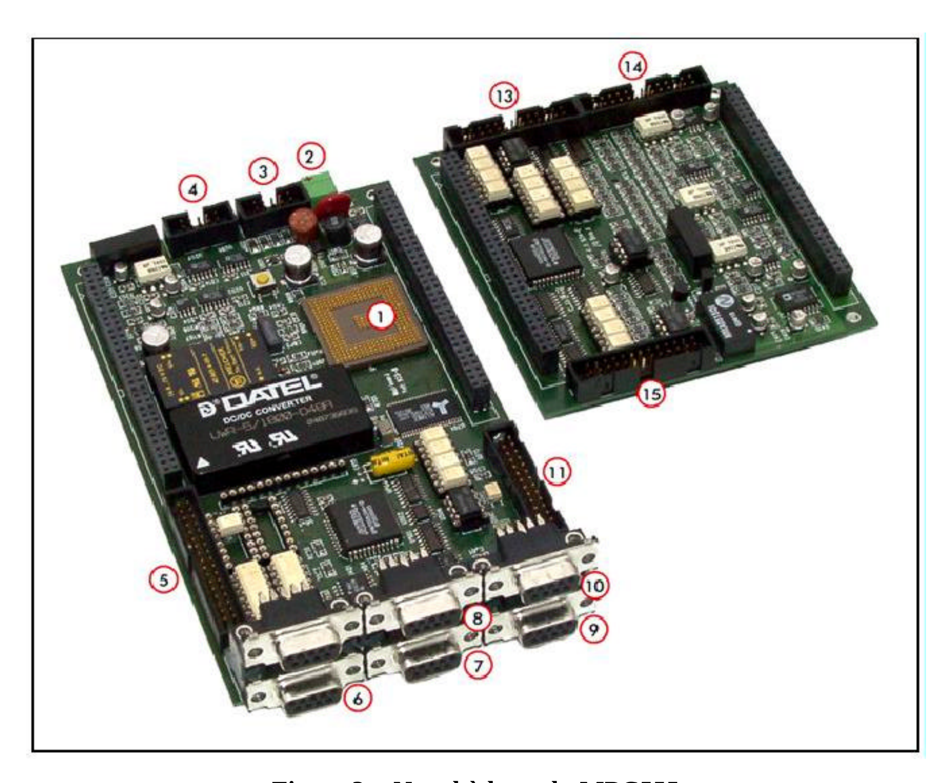


Figure 3 – Nœud à base de MPC555

Figure label	Description
1	MPC555
2	Power supply: 18-60V DC (from batteries)
3	BDM Interface (Basic Debug Interface)
4	7 analog inputs
5	16 logical inputs and 20 logical outputs
6	Synchronous serial line (SPI)
7	Asynchronous serial lines (port 0)
8	Asynchronous serial lines (port 1)
9	CAN bus (port 0)
10	CAN bus (port 1)
from 11 to 14	4 connectors dedicated to axis control (including 1 analog output per axis)

Figure 4 – Les éléments du nœud

2-3 SUJET DU STAGE :

Le sujet de mon stage s'intitule « Programmation des asservissements et de la communication CAN sur le Cycab ». Le but du stage est de transférer les asservissements et la communication CAN (Controller Area Network) conçus pour le Cycab précédent sur ce nouveau Cycab.

3 - Description de l'Implémentation Logicielle sur le CYCAB Rhône-Alpes :

3-1 LE BUS CAN :

Le bus CAN est un bus «série, asynchrone, à 2 fils, symétrique ». Il se présente sous la forme normalisée des différentes couches ISO/OSI. Il en est à sa version 2.0 qui comprend 2 parties A et B. La partie A décrit la trame CAN la plus courante qui permet d'adresser des identificateurs sur 11 bits (contre 29 bits pour la version B ou étendue). Le bus CAN possède les principales propriétés suivantes :

- hiérarchisation des messages,
- garantie des temps de latence,
- souplesse de configuration,
- réception de multiples sources avec synchronisation temporelle,
- système multimaître, détection et signalisation d'erreurs,
- retransmission automatique des messages altérés dès que le bus est libéré,
- distinction entre des erreurs temporaires et des non-fonctionnalités permanentes au niveau d'un nœud,
- déconnexion automatique des nœuds défectueux.

Le protocole CAN a été subdivisé en différentes couches : la couche Objet ; la couche Transfert et la couche Physique.

La couche Objet a pour mission principale de trouver quel message doit être transmis, de décider lequel des messages reçus via la couche transfert est en cours d'utilisation, de produire une interface à la couche applicative en relation avec le « hard » relatif au système.

La couche Transfert a pour mission principale de s'occuper du transfert du protocole. C'est à dire de gérer et de contrôler la mise en forme de la trame, de réaliser l'arbitrage des conflits de bus, de vérifier l'absence ou la présence d'erreurs et de signaler les différents types d'erreurs s'il y en a ainsi que les fautes de confinement. C'est à l'intérieur de cette couche qu'il est décidé si le bus est libre pour démarrer une nouvelle transmission ou bien si une réception d'un message incident est juste en train de démarrer.

La couche Physique a pour rôle d'assurer le transfert physique des bits entre les nœuds en accord avec toutes les propriétés électriques (ou électroniques) du système.

3-2 RAPPEL SUR LE HARDWARE DU CYCAB RHONE-ALPES :

a) Le Cycab possède :

- un PC sous Linux, un module IP-CAN (Controller Area Network), et une liaison ethernet RF (Radio Fréquence).
- deux Modules (traction avant, traction arrière) composés chacun d'un microprocesseur 555 et d'une interface CAN.

- des entités peuvent être rapportées sur le bus CAN (ultrasons, caméra linéaire)

b) Architecture logicielle d'un Module :

A l'amorçage du micro-contrôleur, quatre programmes se déroulent en concurrence :

- le programme principal (main), attend les messages CAN et les traite.
- une routine d'interruption déclenchée sur un TIMER à 10 millisecondes, permet de dérouler des fonctions (périodiques). Celles-ci peuvent être autorisées ou non dynamiquement. A titre d'exemple, sur chaque Module de traction se trouve une routine TIMER qui annule la sortie puissance si celle-ci n'est pas mise à jour régulièrement. Sur le Module Direction-Joystick, une routine TIMER permet l'asservissement en position de la direction.

Il est à noter que le protocole de plus haut niveau, décrit par la suite, (un seul maître, acquittement permanent) permet d'éviter tout blocage sur le MPC555.

3-3 LE PROTOCOLE CAN-CYCAB :

a) Choix sur la communication :

Quelques règles générales...

- le bus CAN est le seul moyen de communication entre la carte et les Modules.
- la carte sous Linux dialogue avec le réseau (pour l'amorçage par exemple) via le lien ethernet RF.

Du point de vue du bus CAN, la carte est maître, alors que les Modules (ou les autres entités CAN) sont esclaves. C'est donc la carte qui prend l'initiative de la communication, le ou les Modules se contentent de rester à l'écoute et de répondre si on leur demande.

b) Le Protocole CAN-CYCAB :

Chaque entité sur le bus CAN doit se conformer au protocole suivant. (**figure 5**)
En résumant, les transferts sur le bus CAN sont basés sur des trames, constituées de 11 bits d'en-tête et de 8 octets de message.
L'entête est utilisé pour identifier le destinataire de manière unique (8 bits) et le type de message (3 bits).
Les différents types de messages sont :

- envoi d'une requête, avec ou sans donnés et avec ou sans demande d'acquittement,
- réponse à une requête, avec ou sans donnés.

Les 8 octets de messages se décomposent en :

- un octet pour l'identifiant de l'envoyeur,
- un octet pour l'identifiant du message,
- un octet pour le mot de commande à exécuter,
- quatre octets pour les données éventuelles,
- dernier octet disponible jamais utilisé.

Dans la version actuelle du CYCAB, seules des requêtes avec demande d'acquittement sont utilisées.

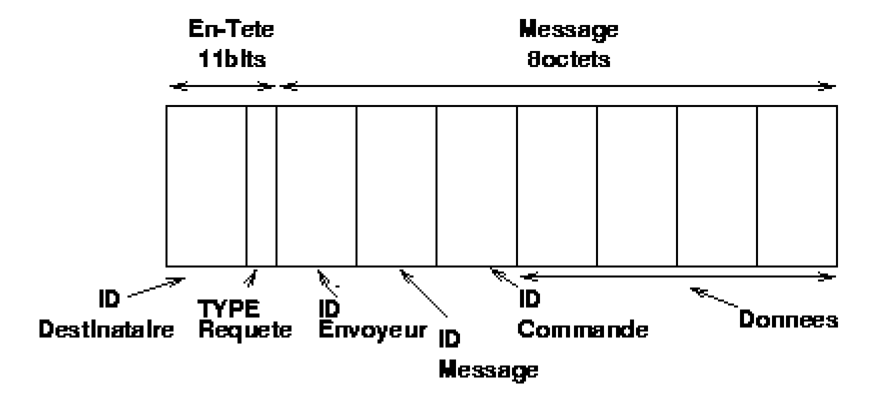


Figure 5 – Trame d'un message

3-4 OUTILS DE PROGRAMMATION :

a) Le compilateur :

Pour le PC sous Linux, la chaîne de développement **GNU** est utilisée. On utilise le compilateur **Cross GCC** (le **GNU** Compiler Collection) un compilateur standard dans le monde UNIX.

La partie **GNU** est aussi utilisée pour développer le code des modules.

b) Les outils Robosoft :

Les outils Robosoft consistent en deux programmes. L'un **elf2sdxbin** permet de transformer le code compilé (format elf) en binaire téléchargeable au MPC555. On peut alors charger ce code sur la carte en utilisant le second programme : **dwnbin**. Notons que le programme **dwnbin** utilise directement les ports d'entrée/sortie de la carte CAN. Il est donc nécessaire d'être ***root*** pour l'utiliser.

4 – Description du code avant le stage :

4-1 LE CODE ROBOSOFT EN LANGAGE ASSEMBLEUR :

Dans le code utilisé sur la version précédente du Cycab, on retrouve une bibliothèque Robosoft écrite en langage assembleur de chez Motorola permettant l'utilisation du MPC555. J'ai donc commencé par étudier le code de la bibliothèque Robosoft concernant la communication via le bus CAN. Ce fut mon premier contact avec de l'assembleur Power-PC.

J'aurai pu traduire ces fonctions en langage C, mais nous avons préféré conserver ces fonctions. C'est à partir des ces fonctions que j'ai pu comprendre et utiliser les fonctions en langage C développées par l'INRIA.

4-2 LE CODE EN C DEVELOPPE PAR L'INRIA :

Pour la version précédente du Cycab, qui fonctionnait sous un environnement VxWorks, l'INRIA avait développé des fonctions en langage C pour la programmation des asservissements et de la communication CAN. Là encore c'est à partir de ces fonctions que j'ai pu comprendre la philosophie du Cycab, et c'est à partir de ces fonctions que j'ai pu construire une architecture logicielle pour le nouveau Cycab.

5 – Description du travail réalisé :

5-1 ARBORESCENCE AUTOUR DU CYCAB :

Mon stage a donc commencé ainsi en établissant une architecture logicielle. J'ai donc défini une arborescence fonctionnelle pour ranger les différents programmes qui gèrent la communication CAN et les asservissements sur le Cycab.

a) Schéma de l'arborescence :

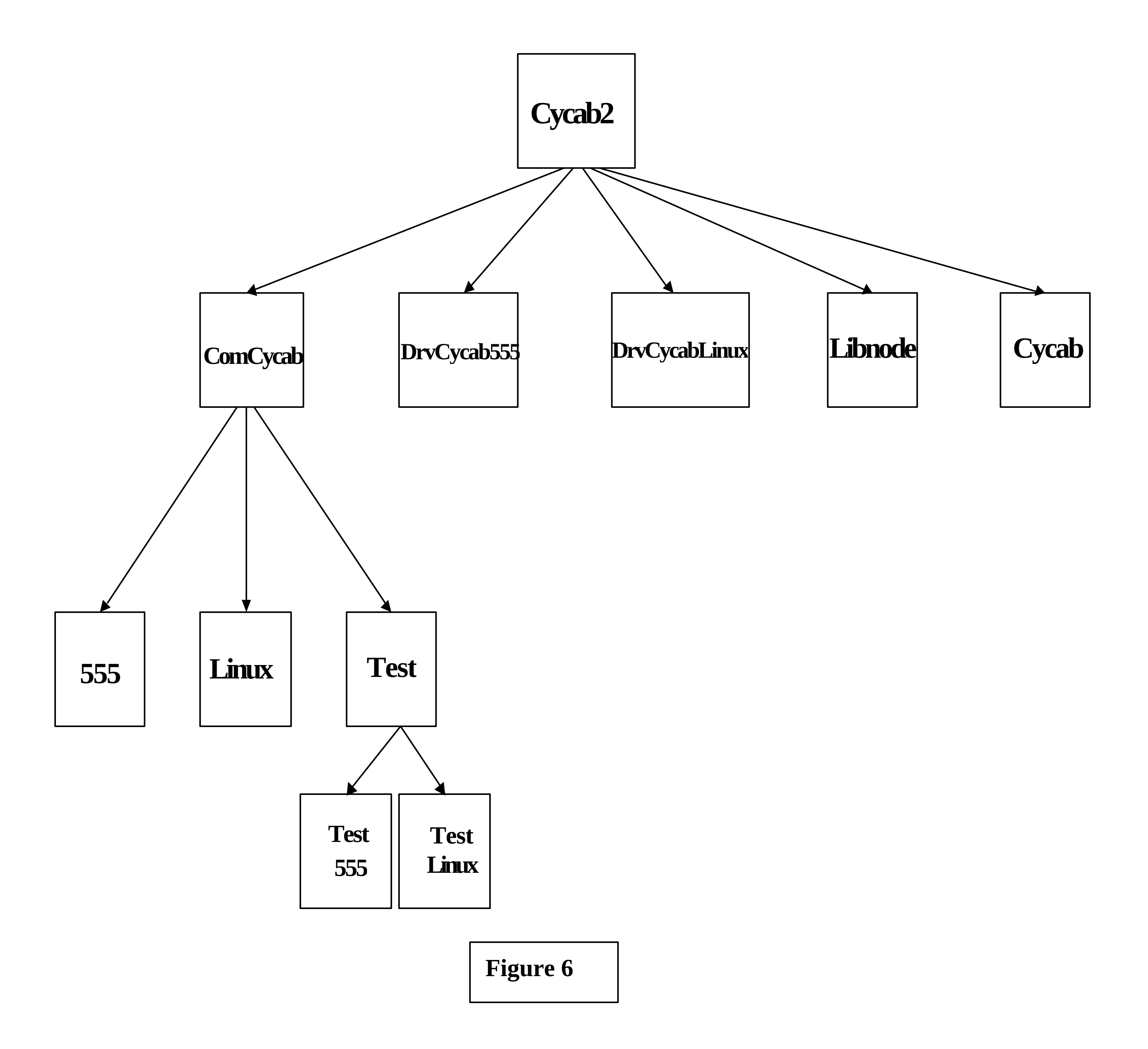


Figure 6

b) Description de l'architecture logicielle du Cycab :

Voici une description des cinq sous-répertoires présents sous **cycab2/**, mais pour une meilleure compréhension il est souhaitable de se référer aux annexes en fin de rapport.

Dans **cycab2/Cycab/** on retrouve toutes les fonctions développées par Robosoft qui permettent à l'utilisateur du Cycab de se servir du véhicule lorsque celui-ci est utilisé dans un mode spécifique. Au démarrage du Cycab, on peut, suivant l'orientation du joystick, choisir cinq modes de fonctionnements. Les modules bootent alors sur le code écrit, par Robosoft, dans leur mémoire flash. Rappelons que le but du stage est d'adapter le propre code développé par l'INRIA au nouveau Cycab, j'ai en fait juste introduit ces fonctions à titre indicatif.

Dans **cycab2/Libnode/** on retrouve les fonctions Robosoft, écrites en langage assembleur de chez motorola, qui sont en fait les fonctions de base du MPC555. Le makefile de ce répertoire génère la librairie **libnode555.a**, qui est utilisée dans la compilation de toutes les fonctions qui sont destinées aux nœuds.

Dans **cycab2/DrvCycabLinux/** on retrouve les fonctions développes par l'INRIA pour gérer les asservissements sur le Cycab, coté utilisateur. Ce sont les pilotes de l'asservissement coté Linux, ils sont utilisables avec l'exécutable ***drvCycabLinux.***

Dans **cycab2/DrvCycab555/** on retrouve les fonctions développées par l'INRIA pour gérer les asservissements sur le Cycab. Ce sont les pilotes cotés MPC555. Le makefile génère un binaire exécutable ***drvCycab555*** qui permet de tester les pilotes des asservissements. Ce binaire est compilé à partir des fichiers **main.c**, **interruption.c** et de **driver.h** :

- Dans ***driver.h*** on a la définition de la structure **MODULE_COM**, en langage C, qui regroupe les paramètres globaux du code. Il s'agit d'une structure assez lourde, mais qui permet de manipuler tous les éléments nécessaires au contrôle du Cycab, dont les routines d'interruptions. Ces interruptions sont gérées dans la structure **MODULE_IT_TIMER**, qui est une variable de la structure **MODULE_COM**.
- Dans ***interruption.c*** on retrouve la fonction ***cycabIntrInit*** qui initialise le tableau **it_timer[5]** de structure **MODULE_IT_TIMER** servant aux fonctions d'interruptions et fixe la fréquence des interruptions grâce à la fonction ***initinterrupt()***. On a ainsi cinq fonctions qui décrivent les routines d'interruption, comme par exemple ***cycabIntrVelocity*** qui s'occupe de réguler la vitesse du Cycab par rapport à une consigne, à chaque fois que cette fonction est appelée. Une de ces fonctions est appelée lorsque sa variable ***status*** est à **OK**, c'est à dire quand l'utilisateur décide de rentre active cette routine d'interruption, et que le compteur des interruptions à la valeur de sa variable ***modulo***, qui fixe la fréquence de cette interruption. Ce sont les fonctions ***cycabIntrStop*** et ***cycabIntrStart***, qui gèrent le passage du ***status*** à **OK** (ce qui active l'interruption) et à **ERROR** (ce qui rend inactive l'interruption).
- Dans ***main.c,*** la fonction ***cycabInit()*** initialise les premières valeurs de la structure **MODULE_COM**. La fonction ***cycabExecute*** dont l'argument d'entrée correspond au message reçu, donc à l'instruction, s'occupe d'appliquer la commande et de renvoyer un accusé de réception au PC.

Dans **cycab2/ComCycab/** j'ai créé le programme ***comCycab.h*** où l'on retrouve les déclarations des structures **CYCAB** et **CanMsgStruct**, en langage C, ainsi que la déclaration des fonctions qui gèrent la communication CAN, côté Linux et MPC555 (dans **cycab2/Comcycab/555/*comCycab555.c*** et **cycab2/Comcycab/Linux/*comCycabLinux.c***).

Mais également trois sous-répertoires :

Le fichier **cycab2/ComCycab/555/** contient les pilotes de la communication, côté 555, et le makefile de ce répertoire génère la librairie ***libcomCycab555.a*** à partir du programme ***comCycab555.c***, qui regroupe les fonctions qui gèrent la communication sur le bus CAN et de ***xFunBasS.h*** où l'on retrouve la déclaration de toutes les fonctions en assembleur du répertoire **cycab2/Libnode/.**

Le fichier **cycab2/ComCycab/Linux/** contient les pilotes de la communication, côté Linux, et le makefile de ce répertoire génère la librairie ***libRcomCycab.a*** à partir du programme ***comCycabLinux.c***, qui regroupe les fonctions qui gèrent la communication sur le bus CAN, de ***robosoft.c*** qui contient les fonctions permettant d'envoyer et de recevoir des trames sur le bus CAN**,** et de ***robosoft.h*** où l'on retrouve la déclaration de toutes les fonctions en C du programme ***comCycabLinux.c.***

Le fichier **cycab2/ComCycab/Test/** contient lui aussi deux sous-réperetoires :

On a ainsi le répertoire **cycab2/ComCycab/Test/Test.555/** où le makefile génère un binaire exécutable qui permet de tester les pilotes de la communication écris dans **cycab2/ComCycab/555/.** Ce binaire est compilé à partir du fichier **test.555.c**, qui est un petit programme test en C avec un main(), et des librairies générées dans **cycab2/ComCycab/555/**. L'exécutable est ensuite transformé au format binaire pour être téléchargé au nœud.

Côté linux, on a le répertoire **cycab2/ComCycab/Test/Test.Linux/** où le makefile génère un binaire exécutable qui permet de tester les pilotes de la communication écris dans **cycab2/ComCycab/Linux/.** Ce binaire est compilé à partir du fichier **test.linux.c**, qui est un petit programme test en C avec un programme principal, et des librairies générées dans **cycab2/ComCycab/Linux**. L'exécutable est ensuite utilisable sous Linux à partir de la commande ***test.linux***.

c) Etapes de la communication entre le PC et le nœud via le bus CAN :

Voici sur un exemple comment se déroule la communication sur le cycab :

Un utilisateur du Cycab décide d'envoyer une commande pour démarrer les moteurs avant. En lançant l'application ***drvCycabLinux*** sur le PC embarqué, il choisit dans le menu d'appliquer cette commande.

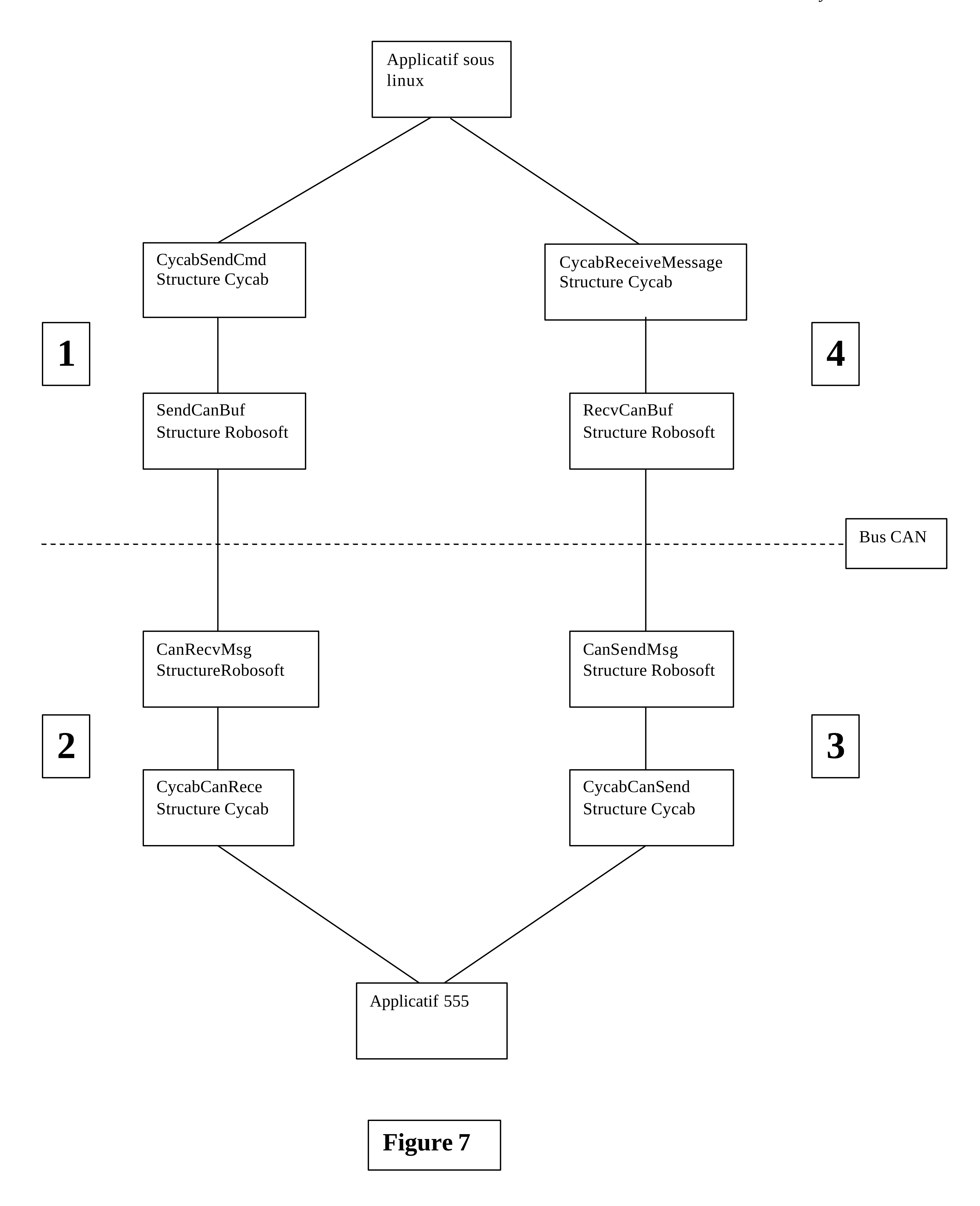

Applicatif sous linux
CycabSendCmd
Structure Cycab
CycabReceiveMessage
Structure Cycab
1
4
SendCanBuf
Structure Robosoft
RecvCanBuf
Structure Robosoft
Bus CAN
CanRecvMsg
StructureRobosoft
CanSendMsg
Structure Robosoft
2
3
CycabCanRece
Structure Cycab
CycabCanSend
Structure Cycab
Applicatif 555

Figure 7

1. Le PC a alors les informations pour construire un message en remplissant la structure **CYCAB.** Le message est envoyé sur le bus CAN par la fonction ***CycabSendCmd***, qui transforme le message en une structure **CanMsgStruct** pour utiliser la fonction ***SendCanBuf*** de ***robosoft.c***.
2. Côté MPC555, le message de structure **CanMsgStruct** est récupéré sur le bus par la fonction Robosoft ***CanRecvMsg*** de **cycab2/Libnode/*canasm.s.*** Le nœud exploite alors le message avec la fonction ***CycabCanRece*** qui assure le passage d'une structure **CanMsgStruct** en une structure **CYCAB**, que le code chargé au nœud pourra comprendre**.**
3. Comme le protocole de communication l'a défini, le nœud doit renvoyer un message pour informer le PC de sa bonne réception. C'est la fonction ***CycabCanSend*** qui transforme le message de structure **CYCAB** en structure **CanMsgStruct,** que la fonction Robosoft ***CanSendMsg*** envoie sur le bus CAN.
4. L'accusé de réception est récupéré alors par le PC avec la fonction ***CycabReceiveMessage***, qui transforme le message en structure **CYCAB**, le message récupéré en structure **CanMsgStruct** par la fonction ***RecvCanBuf*** de ***robosft.c***.

5-2 LA COMMUNICATION SUR UNE CARTE TEST :

Le Service Robotique dispose d'une carte d'évaluation MPC555 (**figure 8**), alimenté par une alimentation 48V comme sur le Cycab et relié via un bus CAN à ***chouffe***, un PC sous Linux qui possède un port CAN. Cette carte possède également un écran LCD qui peut afficher des caractères à l'aide de la fonction ***LCDPutString()*** dans les programmes compilés et téléchargés MPC555.

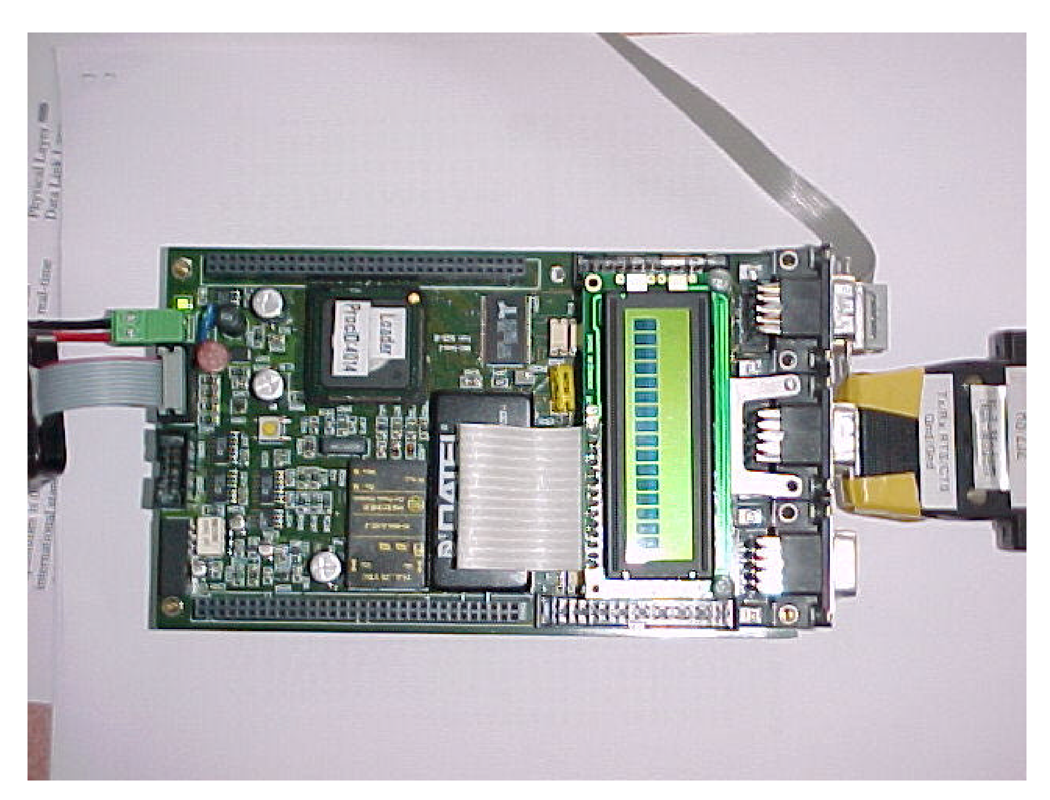


Figure 8 – Carte Test

Ce MPC555 possède un code minimum chargé dans sa mémoire flash, si bien qu'au démarrage, à la mise sous tension donc, cette carte se place en attente de recevoir du code. Après chaque reset, on doit donc recharger du code au MPC555 par l'intermédiaire du bus CAN. Pour identifier le destinataire du code dans le réseau CAN, la carte possède un numéro d'identifiant en hexadécimal. Ici 4014 qui correspond à 16404 en décimal. Pour tester la communication sur le bus CAN, on se place root sur ***chouffe***, on se met dans le répertoire **cycab2/ComCycab/Test/Test.555/*test.555.c*** et on télécharge alors le programme exécutable à partir de la commande ***dwnbin test.555.bin 16404***, où ***test.555.bin*** est un exécutable compilé sous Linux et transformé en format binaire. Notons qu'une fois le code téléchargé, le MPC555 boote directement sur ce code et commence son exécution. Pour valider la communication sur le bus CAN, on doit également lancer en parallèle le programme test exécutable qui tourne sous Linux. On se place là aussi root sur ***chouffe***, on se met dans le répertoire **cycab2/ComCycab/Test/Test.linux/*test.linux.c*** et on lance le programme à partir de la commande ***test.linux***.

a) Première Série de Tests :

Dans les premiers tests de communication effectués, nous avions choisit de commencer les échanges par un message qui serait écrit en dur dans le programme test téléchargé et envoyé par le MPC555 sur le bus CAN, alors que Linux, lui, de son coté restait en attente d'un message. On décide donc de valider le passage de l'étape 3 à l'étape 4 du schéma de la **figure 7**. Les premiers tests réalisés avec différentes valeurs pour les 11 bits consacrés à l'identifiant retournaient des messages avec toujours les mêmes valeurs d'identifiant. En fait tout se passait comme si les 11 bits étaient tous à zéro.

Il a donc fallu reprendre le chemin séquentiel suivi par la structure **CYCAB**. On retrouvait dans le main du programme **test.555.c**, une structure **CYCAB** écrite en dure, c'est à dire dont les valeurs avaient été fixées, et qui était alors envoyée vers le PC sous Linux par la fonction ***cycabCanSend.*** Cette fonction, qui transforme cette structure **CYCAB** en structure Robosoft, fait appel à la fonction ***CANSendMsg*** décrite en assembleur, dans le programme ***CANASM.s*** du répertoire **cycab2/Libnode/**.

En regardant le code assembleur de cette fonction, je me suis aperçu que lors de la phase de manipulation des registres, celui qui contenait la valeur de l'identifiant était mal exploité puisque seuls les bits de 16 à 23, sur les 32 bits du registre, étaient utilisés dans la suite alors que cet emplacement est toujours vide (bits à zéro) !

J'ai donc réalisé un nouveau masque sur ce registre qui nous garantissait de bien retrouver les 11 bits qui codent pour l'identifiant du message.

Mais malgré cette modification apportée au code assembleur de Robosoft, les tests continuaient à présenter les mêmes résultats, c'est à dire un identifiant nul.

Il a valut continuer à remonter le code pour voir ou se situait l'erreur.

Coté Linux, on était en attente d'un message grâce à une boucle infinie qui appelait la fonction ***cycabReceiveMessage*** qui reçoit un message du CAN grâce à la fonction ***recvCANbuf*** décrite dans le programme ***Robosoft.c*** du répertoire **cycab2/Comcycab/Linux/** et qui va lire sur le bus CAN un éventuel message.

En regardant de plus près le code C de cette fonction, j'ai vu que la variable **adr** qui doit recevoir la valeur de l'identifiant du message était mal manipulée. En effet les bits 3 à 10 était rangés dans **adr** auquel on ajoutait avec l'opérateur & (opérateur logique et) les bits 0 à 2, si bien qu'au final on se retrouvait avec une valeur nulle dans **adr**, soit tous les bits de l'identifiant à zéro.

J'ai donc remplacé l'opérateur & par un opérateur | (opérateur logique ou) pour retrouver toutes les valeurs de l'identifiant sans perdre de bits.

Une fois ces modifications apportées, les bibliothèques régénérées, c'est à dire après compilation des pilotes de la communication des répertoires **cycab2/ComCycab/555/** et **cycab2/ComCycab/linux/,** les exécutables recompilés et le code du MPC555 téléchargé, j'ai pu reprendre les tests et vérifier que le PC sous Linux affichait bien les valeurs attendues pour l'identifiant, c'est à dire les valeurs écrites en dures dans le programme **cycab2/ComCycab/Test/Test.555/test.555.c**.

b) <u>Deuxième Série de Tests :</u>

Pour la deuxième série de tests, on voulait vérifier le circuit total emprunté par les messages dans le cadre du fonctionnement sur le Cycab. On décide donc de valider le passage de l'étape 1 à l'étape 4 du schéma de la **figure 7**.

Pour cela le PC sous Linux devait envoyer un message écrit en dur dans le programme test **cycab2/ComCycab/Test/Test.linux/test.linux.c**, alors que le MPC555 passait en attente de message. Une fois le message reçu, celui-ci était renvoyé aussitôt sur le bus CAN, puis reçu par le PC sous Linux qui était repassé en mode attente de message. Mais j'avais laissé dans ce nouveau test, le test précédent que je venais de valider.

Le début du programme se déroulait bien, à savoir, le message écrit coté MPC555, puis envoyé, était bien reçu par Linux, qui ensuite envoyait un nouveau message qui ne devait faire qu'un aller retour entre lui et le MPC555. Mais le second message reçu par Linux (celui qui devait faire l'aller-retour) correspondait en réalité au premier message écrit et envoyé par le MPC555 au début. Tout se passait comme si le message écrit en dur coté MPC555 n'était pas écrasé par les messages envoyés par Linux.

En fait le problème se trouvait dans le registre **I_FLAG** de la fonction ***canasm.s***, qui contient un bit qui est passé à 1 pendant une réception/transmission et qui est remis à zéro à la fin d'une bonne réception/transmission.

Il a fallut passer outre, et même si dans les tests où l'on commençait par envoyer des messages de Linux, c'est à dire sans écrire en dur coté MPC555, on ne retrouvait pas ce problème. Pour cela et se prévenir d'éventuels problèmes dans les futurs tests, nous avons du rajouter des vérifications de conditions par les tests de communications. On a donc introduit un marqueur sur chaque message qui vérifie pour chaque réception la valeur de l'identifiant en regardant la variable **can_in.id**.

A ce moment les tests de communication répondaient bien au cahier des charges qui avait été choisi par le service robotique. Mais dans le but d'implémenter ce protocole de communication au nouveau Cycab qui est composé de deux nœuds, on a choisi d'introduire

un argument dans le programme exécutable de Linux pour distinguer les différents nœuds avec lesquelles le PC peut entrer en communication. On aura bien sur auparavant écrit en dur dans le programme à télécharger à chaque nœud l'identifiant de ce dernier (concrètement on choisit le numéro 1 pour le nœud avant et 2 pour l'arrière). Désormais chaque module possède un identifiant unique, et donc un fichier généré unique, puisqu'il a été décidé de ne pas avoir d'allocation dynamique d'identifiant.

Pour améliorer le test on rajoute un deuxième argument pour que l'utilisateur fixe le nombre de messages envoyés par le PC en direction d'un nœud.

Une fois cette modification apportée, les bibliothèques régénérées, les exécutables recompilés et le code du MPC555 téléchargé, j'ai pu reprendre les tests et vérifier que le PC sous Linux affichait bien les valeurs attendues, c'est à dire les valeurs écrites en dures dans le programme **cycab2/ComCycab/Test/Test.linux/test.linux.c**.

On valide alors définitivement la communication en téléchargeant au nœud le binaire exécutable recompilé avec la valeur 1 pour le numéro d'identifiant sur le bus CAN avec toujours la même commande ***dwnbin test.555.bin 16404***. Puis coté PC, on lance la commande ***test.linux 1 6*** :

On a alors à l'écran :

```
cycabComInit...
init du device can sur le PC linux
 work on CAN 0 = d800
adr: rec(0x1), ra(req), dp(dp), ack(ack)
trame: exp(0x0), id(0x0), key(0x33)
data: 0x0 0x0 0x0 0x0
valeur = 0x0
adr: rec(0x1), ra(req), dp(dp), ack(ack)
trame: exp(0x0), id(0x1), key(0x33)
data: 0x0 0x0 0x0 0x1
valeur = 0x1
adr: rec(0x1), ra(req), dp(dp), ack(ack)
trame: exp(0x0), id(0x2), key(0x33)
data: 0x0 0x0 0x0 0x2
valeur = 0x2
adr: rec(0x1), ra(req), dp(dp), ack(ack)
trame: exp(0x0), id(0x3), key(0x1)
data: 0x0 0x0 0x0 0x0
reponse = 0
adr: rec(0x1), ra(req), dp(dp), ack(ack)
trame: exp(0x0), id(0x4), key(0x1)
data: 0x0 0x0 0x0 0x1
reponse = 1
adr: rec(0x1), ra(req), dp(dp), ack(ack)
trame: exp(0x0), id(0x5), key(0x1)
data: 0x0 0x0 0x0 0x2
reponse = 2
adr: rec(0x1), ra(req), dp(no_dp), ack(ack)
trame: exp(0x0), id(0x6), key(0xFF)
```

```
cycabComClose...
```

5-3 LE PROBLEME DES INTERRUPTIONS :

Pour valider les tests d'interruptions on doit téléchargé au nœud l'exécutable ***drvCycab555*** généré par **cycab2/DrvCycab555/.** Dans la fonction ***main()*** qui fait appel à la fonction ***cycabInit()*** qui doit, elle, initialiser la structure **CYCAB_COM**, on fait appel à la fonction ***InitInterrupt(int PITValue)*** qui fixe pour le MPC555 la fréquence des routines d'interruptions en choisissant PITValue à l'aide de la formule :

PITPériode = (PITValue + 1)/15625

Où PITPeriod est exprimée en secondes.

En fait, on fixe ici la fréquence à laquelle, la fonction ***interruptHandlerPIT()*** va être appelée. A partir de ce moment le MPC555 prend le contrôle des opérations, on sort du mode séquentiel, désormais des interruptions sont générées et on effectue alors à chaque période d'interruption la fonction ***interruptHandlerPIT().***

Cette fonction contient un compteur qui gère le nombre de passage dans ***interruptHandlerPIT(),*** donc le nombre de passage depuis la dernière interruption. Ensuite, on vérifie, dans cette fonction, pour toutes les procédures d'interruption la valeur du ***status*** ainsi que la valeur du compteur par rapport à celle du ***modulo***.

Pour tester les interruptions on charge au nœud l'exécutable ***drvCycab555***, généré à partir du programme ***interruption.c*** notamment. Dans ce programme on crée une routine d'interruption test qui ne contient qu'un simple compteur de boucle pour contrôler le nombre de passage dans cette routine. On ne passe la variable ***status*** à ***OK*** que pour cette routine.

Lors des premiers tests, le LCD affiche que le MPC555 est bien en attende d'un message de Linux. Mais une fois un message envoyé, lorsque l'on recharge le code, après avoir reseter, on ne retourne plus dans la boucle d'attente. En fait dans le manuel d'utilisateur du micro-contrôleur on apprend que le bus CAN lui-même génère des interruptions.

On s'assure alors que le bus CAN ne génère pas d'interruptions en vérifiant dans **cycab2/Libnode/*CANASM.s*** que les masques ne déclenchent pas les différentes routines d'interruptions possibles :

Premier type d'interruptions :

Le registre **IMASK** détermine quel buffer (c'est là que sont stockés les messages reçus où envoyés sur le bus CAN) génère une interruption (si le bit correspondant est mis à 1). On a du modifier la valeur de ce registre, la mettre à 0 ici, pour tester les interruptions, alors qu'en fonctionnement normal le buffer 0 doit générer une interruption après chaque bonne transmission/réception.

Deuxième type d'interruptions :

Il existe également 3 autres interruptions bus off, error et wake up. On s'assure alors que dans le registre **CANCTRLO**, les bits **BOFFMSK** et **ERRMSK** soient bien mis à zéro. Ainsi que le bit **WAKEMSK** dans le registre **TCNMCR**.

Une fois ces régalages effectués, on peut tester les interruptions. On réalise alors une fonction d'interruption test qui contient un compteur dont on affiche la valeur sur l'écran grâce à la fonction ***LCDPutString***, en s'assurant qu'elle ne génère pas d'interruptions !

On vérifie ainsi le bon fonctionnement du changement de la valeur du ***status*** dans la structure **CYCAB_COM** ainsi que la bonne correspondance entre les valeurs du compteur des interruptions et celui de la fonction test effectuée.

On valide donc les routines d'interruptions en vérifiant qu'également que l'on peut faire descendre la période des interruptions jusqu'à **1 ms**, lorsque les routines d'interruptions sont vides, c'est à dire avec juste un compteur.

5-4 LA COMMUNICATION SUR LE CYCAB :

Une fois les différents tests de communication et d'interruptions validés, j'ai pu passer aux essais sur le nouveau Cycab, dit « Cycab blanc ». Jusqu'à présent personne n'avait essayé de charger du code au nouveau Cycab. Le PC sous linux embarqué sur le Cycab s'appelle ***bcycab***.

Lorsque l'on met le Cycab sous tension, les deux MPC555 bootent alors sur leur mémoire flash qui contient le code écrit par Robosoft, où l'on retrouve notamment la structure Robosoft que l'on avait repris pour programmer la communication.

Pour valider le test d'asservissement du répertoire **cycab2/Drvcycab.555/** on doit télécharger au nœud le programme exécutable, grâce au bus CAN, en spécifiant pour chaque nœud sont numéro d'identifiant. (Ici, le nœud avant a comme ID, en Héxa ; 0x4009, soit, en décimal, ID=16393).

On se met root sous ***bcycab*** et on tape la commande ***dwnbin DrvCycab555.bin 16393***, mais le transfert s'arrête après l'envoie de la première trame de 8 bytes, sans afficher l'accusé de réception. On ne peut donc pas envoyer directement du code nouveau au MPC555. Il faut absolument le faire entrer en mode Loader.

En regardant le code Robosoft de plus prêt, dans le programme ***cycab2/cycab/000MAI_C.c***, on se rend compte que l'on peut faire passer le MPC555 en mode Loader, ce qui est possible en lui envoyant un message comportant les valeurs suivantes, à partir de la structure Robosoft :

Trame[0]=18*256, Trame[1]=0, Trame[2]= 3, et pour i= 3,..,7, Trame[i]=0.

Premier Test - Trame Loader sur *bcycab* :

On se met root sous ***bcycab*** et on envoie par Linux un message qui doit permettre au MPC555 de se mettre en mode Loader. On utilise alors la fonction ***cycabSendCmd*** qui fait appelle à la fonction ***sendCanBuf*** de Robosoft. Pour suivre ce message, on demande à Linux d'afficher le message de retour, mais dans le suivi séquentiel du programme principal, on en conclut que le message est bien envoyé, mais que celui-ci n'est pas renvoyé. De plus, lorsque l'on essaye de télécharger au MPC555 l'exécutable, on se retrouve dans le même cas que lors du premier test, à savoir, bloquer après l'envoie de la première trame de 8 bytes.

Ceci nous laisse alors penser à un problème d'identifiant, c'est à dire que le message envoyé sur le Bus CAN n'est en fait jamais lu par le MPC555.

Remarque : On réalise ces essais en testant les 2 ports du CAN. (CAN0 et CAN1).

Deuxième Test - Test du PC :

On décide alors de vérifier le bon fonctionnement du PC Linux embarqué. Pour cela, on branche sur le PC le bus CAN et le MPC555 qui m'avait servi à tester la communication, puisque ce MPC555 ne boote pas sur un programme écrit dans sa mémoire flash. Avec la commande ***dwnbin test.555.bin 16404*** on envoie au MPC555 le code du programme test, mais là encore on n'arrive pas à télécharger du code.

En reprenant le rapport de Fabien LYDOIRE on se rend compte qu'il faut adapter le programme ***dwnbin*** aux ports d'entrées/sorties de la carte CAN, valeurs que l'on obtient en effectuant la commande ***cat /proc/pci.*** Dans notre cas on doit adapter ***dwnbin*** à l'adresse du CAN sur ***bcycab***.

On remplace donc dans le code de ***dwnbin.c***

```
#define CAN0 0xD800
```

par

```
#define CAN0 0xE000
```

Ce changement doit être réalisé dans le programme ***dwnbin.c*** dont le makefile génère un exécutable (c'est à dire la commande) ***dwnbin_bcycab*** qui permettra de télécharger du code aux nœuds qui sont sur le bus CAN de ***bcycab.*** Mais également dans le programme ***robosoft.c*** du répertoire ***cycab2/Comcycab/Linux/*** qui gère la communication côté Linux et notamment la génération d'une librairie. On compile et on obtient un exécutable ***test.linux_bcycab*** pour le ***bcycab***.

On se met root sur bcycab, on télécharge le code test au MPC555 avec la commande ***dwnbin_bcycab test.555.bin 16404***, on vérifie que le code est bien chargé et que le LCD affiche bien le fait que le MPC555 soit en attente de messages. Puis on tape la commande ***test.linux_bcycab*** pour valider le test. On obtient alors les mêmes résultats que pour le test effectué sur ***chouffe***.

On vérifie ainsi le bon fonctionnement du PC Linux embarqué sur ***bcycab.***

On essaye donc de nouveau de faire passer le MPC555 en mode Loader en se plaçant root sur ***bcycab*** avec la commande ***test.loader_bcycab***, le programme ayant été recompilé avec les changements d'adresse du CAN apportés dans ***robosoft.c.*** Mais là encore, si le message est bien envoyé sur le bus CAN, on ne reçoit jamais de message-réponse de la part du MPC555, qui ne passe toujours pas en mode loader.

Troisième test - Test du câble :

Devant la complexité du problème, j'ai décidé de tester d'autres configurations pour essayer de déterminer l'origine du problème.

On teste alors le bon fonctionnement du câblage CAN du Cycab blanc, en branchant la carte sur le bus CAN du Cycab :

J'ai ainsi branché sur le circuit CAN du Cycab blanc à la place de la carte avant, le MPC555 du test de la communication, puis j'ai essayé de lui télécharger l'exécutable du test de la communication avec la commande ***dwnbin_bcycab test.555.bin 16404***. J'ai réussi à télécharger l'intégralité du code compilé et au format binaire. On teste alors la communication en lançant le programme test du côté Linux avec la commande ***test.linux_bcycab 1 6***, et on retrouve bien en réception les deux séries de réponses avec trois valeurs 0x00, 0x01 et 0x02, puis les trois réponses 0, 1 et 2, comme pour le test sur ***chouffe***.

On obtient alors à l'écran les mêmes résultats que pour le test de communication sur ***chouffe***. On vérifie alors que le bus CAN du Cycab fonctionne correctement.

Quatrième test - Trame Loader sur *chouffe* :

On essaye d'envoyer la trame qui est sensé faire passer la carte en mode loader au nœud avant, on retire donc la carte du Cycab, on la connecte sur ***chouffe***, on se place root et on envoie la trame en question par la commande ***test.loader_chouffe***. Mais on obtient les mêmes messages d'erreur que lors des essais réalisés sur le Cycab.

Cinquième test - Avec SDS :

Le service robotique dispose d'un outil de debug pour le MPC555, il s'agit du logiciel SDS, il permet l'exécution pas à pas d'un programme ainsi que la visualisation de la valeur des registres par le port BDM du module. J'ai donc retiré le nœud arrière du Cycab pour regarder l'état des différents registres gérant l'accès à la mémoire de ce nœud. J'ai ensuite utilisé l'option reset de SDS, en espérant ainsi effacer la mémoire du MPC555. J'ai rebranché ensuite le nœud arrière sur le Cycab et j'ai recommencé les tests, dans le but de télécharger aux nœuds le code développé à l'INRIA.

J'ai donc essayé de lui télécharger l'exécutable du test de la communication avec la commande ***dwnbin_bcycab test.555.bin 16393***, en pensant télécharger le code au nœud avant. Le téléchargement a alors réussi sans rester bloquer après l'envoie de la première trame de 8 byt

es, mais en fait le code envoyé sur le bus CAN était récupéré par le nœud arrière. J'ai pu le vérifier en recommençant le test en débranchant le nœud avant et en donnant une valeur quelconque pour le numéro d'identifiant du module. Il fallait ensuite valider le téléchargement en lançant ***test.linux_bcycab 1 6*** sur le PC embarqué.

On a alors à l'écran :

```
cycabComInit...
init du device can sur le PC linux
 work on CAN 0 = e000
adr: rec(0x1), ra(ack), dp(no_dp), ack(no_ack)
trame: exp(0xA), id(0x0), key(0x0)
data: 0x0 0x0 0x0 0x0
valeur = 0x134521372
```

Alors que l'on aurait du obtenir à l'écran :

```
cycabComInit...
init du device can sur le PC linux
 work on CAN 0 = e000
adr: rec(0x1), ra(req), dp(dp), ack(ack)
trame: exp(0x0), id(0x0), key(0x33)
data: 0x0 0x0 0x0 0x0
valeur = 0x0
```

On ne retrouve donc pas les valeurs écrites en dures dans le programme **cycab2/ComCycab/Test/Test.linux/*test.linux.c***.

Ceci montre que le téléchargement n'a pas réussi. Le transfert des trames s'était pourtant bien effectué, mais à l'évidence il y a un problème dans l'écriture des registres.

C'est arrivé à cette étape que j'ai interrompu mon travail afin de préparer mon rapport de stage. Après cela, je disposerai encore de deux semaines de stages pour achever mon travail. J'essayerai donc à partir de la carte test, que je placerai sur le Cycab, de valider les commandes pour les moteurs.

6- CONCLUSION :

Ce stage m'a donné l'occasion de suivre et comprendre les différentes étapes de développement d'une application de contrôle-commande à base de micro-contrôleur en partant de la carte électronique, en passant par le langage assembleur, pour arriver au langage C. De plus le fait de s'impliquer et de comprendre des programmes déjà existants pour les adapter à un nouvel environnement aura été très formateur. Les nombreux problèmes survenus lors des essais sur le Cycab m'auront appris à prendre des initiatives pour cibler les causes de ces problèmes. Enfin, j'aurai également découvert de nouvelles notions autour des architectures distribuées, grâce notamment à :

- L'étude et la mise en place d'un protocole de communication entre deux identités différentes.
- L'étude et la validation des codes en langage assembleur fournis par Robosoft.

Ce stage m'aura ainsi permis de suivre mon projet de la conception à l'implémentation, d'intégrer des composants matériels et logiciels complexes, tout en travaillant en équipe avec les doctorants et chercheurs utilisant le Cycab.

La documentation technique laissée à l'INRIA sera reprise et complétée dans un rapport technique de l'INRIA. Elle permettra la poursuite et l'utilisation de mon travail ainsi que le développement d'application fonctionnant sur le nœud Robosoft dans le cas du Cycab.

7- BIBLIOGRAPHIE :

- [1] La page web de l'INRIA :
http://www.inria.fr

- [2] La page web de l'INRIA Rhône-Alpes :
http://www.inrialpes.fr

- [3] La page web du Service Robotique, Vision et Réalité Virtuelle de l'INRIA Rhône-Alpes :
http://www.inrialpes.fr/iramr

- [4] La page de Motorola :
http://www.motorola.com

- [5] La page de Robosoft :
http://www.robosoft.fr

- [6] Motorola, « MPC555 User's Manual »

- [7] Motorola, « MPC555 Evaluation Board, Quick Reference »

- [8] Motorola, « Time Processor Unit Reference Manual »

- [9] Rapport de stage DESS Essi, Fabien Lydoire, « Programmation d'une centrale de contrôle-commande à base de Power-PC ».

- [10] Rapport technique, Gérard Baille – Philippe Garnier – Hervé Mathieu – Roger Pissard-Gibollet, « Le Cycab de l'INRIA Rhône-Alpes ».